Stochastic Low-Rank Kernel Learning for Regression

Pierre Machart, Thomas Peel, Liva Ralaivola, Sandrine Anthoine, Hervé Glotin

Stochastic Low-Rank Kernel Learning for Regression

Pierre Machart PIERRE.MACHART@LIF.UNIV-MRS.FR
LIF, LSIS, CNRS, Aix-Marseille Université

Thomas Peel THOMAS.PEEL@LIF.UNIV-MRS.FR
LIF, LATP, CNRS, Aix-Marseille Université

Sandrine Anthoine ANTHOINE@CMI.UNIV-MRS.FR
LATP, CNRS, Aix-Marseille Université

Liva Ralaivola LIVA.RALAIVOLA@LIF.UNIV-MRS.FR
LIF, CNRS, Aix-Marseille Université

Hervé Glotin GLOTIN@UNIV-TLN.FR
LSIS, CNRS, Université du Sud-Toulon-Var

Abstract

We present a novel approach to learn a kernel-based regression function. It is based on the use of conical combinations of data-based parameterized kernels and on a new stochastic convex optimization procedure of which we establish convergence guarantees. The overall learning procedure has the nice properties that a) the learned conical combination is automatically designed to perform the regression task at hand and b) the updates implicated by the optimization procedure are quite inexpensive. In order to shed light on the appositeness of our learning strategy, we present empirical results from experiments conducted on various benchmark datasets.

1. Introduction

Our goal is to learn a kernel-based regression function, tackling at once two problems that commonly arise with kernel methods: working with a kernel tailored to the task at hand *and* efficiently handling problems whose size prevents the Gram matrix from being stored in memory. Though the present work focuses on regression, the material presented here might as well apply to classification.

Compared with similar methods, we introduce two novelties. Firstly, we build conical combinations of rank-1 Nyström approximations, whose weights are chosen so as to serve the regression task – this makes our approach different from (Kumar et al., 2009) and (Suykens et al., 2002), which focus on approximating the full Gram matrix with no concern for any specific learning task. Secondly, to solve the convex optimization problem entailed by our modeling choice, we provide an original stochastic optimization procedure based on (Nesterov, 2010). It has the following characteristics: i) the computations of the updates are inexpensive (thanks to the designing choice of using rank-1 approximations) and ii) the convergence is guaranteed. To assess the practicality and effectiveness of our learning procedure, we conduct a few experiments on benchmark datasets, which allow us to draw positive conclusions on the relevance of our approach.

The paper is organized as follows. Section 2 introduces some notation and our learning setting; in particular the optimization problem we are interested in and the rank-1 parametrization of the kernel our approach builds upon. Section 3 describes our new stochastic optimization procedure, establishes guarantees of convergence and details the computations to be implemented. Section 4 discusses the hyperparameters inherent to our modeling as well as the complexity of the proposed algorithm. Section 5 reports results from numerical simulations on benchmark datasets.

2. Proposed Model

Notation $\mathcal{X}$ is the input space, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ denotes the (positive) kernel function we have at hand and $\phi : \mathcal{X} \to \mathcal{H}$ refers to the mapping $\phi(\boldsymbol{x}) := k(\boldsymbol{x}, \cdot)$ from $\mathcal{X}$ to the reproducing kernel Hilbert space $\mathcal{H}$ associated with k. Hence,

Appearing in *Proceedings of the 28^{th} International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

$k(\boldsymbol{x}, \boldsymbol{x}')=\langle\phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\rangle$, with $\langle\cdot,\cdot\rangle$ the inner product of $\mathcal{H}$.

The training set is $\mathcal{L} := \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \in (\mathcal{X} \times \mathbb{R})^n$, where y_i is the target value associated to $\boldsymbol{x}_i$. $K = (k(\boldsymbol{x}_i, \boldsymbol{x}_j))_{1\leq i,j\leq n} \in \mathbb{R}^{n\times n}$ is the Gram matrix of k with respect to $\mathcal{L}$. For $m = 1, \ldots, n$, $\boldsymbol{c}_m \in \mathbb{R}^n$ is defined as:

$$\boldsymbol{c}_m := \frac{1}{\sqrt{k(\boldsymbol{x}_m, \boldsymbol{x}_m)}}[k(\boldsymbol{x}_1, \boldsymbol{x}_m), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}_m)]^\top.$$

2.1. Data-parameterized Kernels

For $m = 1, \ldots, n$, $\tilde{\phi}_m : \mathcal{X} \to \tilde{\mathcal{H}}_m$ is the mapping:

$$\tilde{\phi}_m(\boldsymbol{x}) := \frac{\langle\phi(\boldsymbol{x}), \phi(\boldsymbol{x}_m)\rangle}{k(\boldsymbol{x}_m, \boldsymbol{x}_m)}\phi(\boldsymbol{x}_m). \tag{1}$$

It directly follows that $\tilde{k}_m$ defined as, $\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$,

$$\tilde{k}_m(\boldsymbol{x}, \boldsymbol{x}') := \langle\tilde{\phi}_m(\boldsymbol{x}), \tilde{\phi}_m(\boldsymbol{x}')\rangle = \frac{k(\boldsymbol{x}, \boldsymbol{x}_m)k(\boldsymbol{x}', \boldsymbol{x}_m)}{k(\boldsymbol{x}_m, \boldsymbol{x}_m)},$$

is indeed a positive kernel. Therefore, these parameterized kernels $\tilde{k}_m$ give rise to a family $(\tilde{K}_m)_{1\leq m\leq n}$ of Gram matrices of the following form:

$$\tilde{K}_m = (\tilde{k}_m(\boldsymbol{x}_i, \boldsymbol{x}_j))_{1\leq i,j\leq n} = \boldsymbol{c}_m\boldsymbol{c}_m^T, \tag{2}$$

which can be seen as rank-1 Nyström approximations of the full Gram matrix K (Drineas & Mahoney, 2005; Williams & Seeger, 2001).

As studied in (Kumar et al., 2009), it is sensible to consider convex combinations of the $\tilde{K}_m$ if they are of very low rank. Building on this idea, we will investigate the use of a parameterized Gram matrix of the form:

$$\tilde{K}(\boldsymbol{\mu}) = \sum_{m\in\mathcal{S}} \mu_m\tilde{K}_m \quad \text{with} \quad \mu_m \geq 0, \tag{3}$$

where $\mathcal{S}$ is a set of indices corresponding to the specific rank-one approximations used. Note that since we consider *conical* combinations of the $\tilde{K}_m$, which are all positive semi-definite, $\tilde{K}(\boldsymbol{\mu})$ is positive semi-definite as well.

Using (1), one can show that the kernel $\tilde{k}_{\boldsymbol{\mu}}$, associated to our parametrized Gram matrix $\tilde{K}(\boldsymbol{\mu})$, is such that:

$$\tilde{k}_{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{x}') = \langle\phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\rangle_A = \phi(\boldsymbol{x})^\top A\phi(\boldsymbol{x}), \tag{4}$$

$$\text{with} \quad A := \sum_{m\in\mathcal{S}} \mu_m \frac{\phi(\boldsymbol{x}_m)\phi(\boldsymbol{x}_m)^\top}{k(\boldsymbol{x}_m, \boldsymbol{x}_m)}. \tag{5}$$

In other words, our parametrization induces a modified metric in the feature space $\mathcal{H}$ associated to k. On a side note, remark that when $\mathcal{S} = \{1 \ldots, n\}$ (i.e. all the columns are picked) and we have uniform weights $\boldsymbol{\mu}$, then $\tilde{K}(\boldsymbol{\mu}) = KK^\top$, which is a matrix encountered when working with the so-called empirical kernel map (Schölkopf et al., 1999).

From now on, M denotes the size of $\mathcal{S}$ and m_0 refers to the number of non-zero components of $\boldsymbol{\mu}$ (i.e. it is the 0-pseudo-norm of $\boldsymbol{\mu}$).

2.2. Kernel Ridge Regression

Kernel Ridge regression (KRR) is the kernelized version of the popular ridge regression (Hoerl & Kennard, 1970) method. The associated optimization problem reads:

$$\min_{\boldsymbol{w}} \left\{ \lambda\|\boldsymbol{w}\|^2 + \sum_{i=1}^n \left(y_i - \langle\boldsymbol{w}, \phi(\boldsymbol{x}_i)\rangle\right)^2 \right\}, \tag{6}$$

where $\lambda > 0$ is a regularization parameter.

Using I for the identity matrix, the following dual formulation may be considered:

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left\{ F_{KRR}(\boldsymbol{\alpha}) := \boldsymbol{y}^T\boldsymbol{\alpha} - \frac{1}{4\lambda}\boldsymbol{\alpha}^T(\lambda I + K)\boldsymbol{\alpha} \right\}. \tag{7}$$

The solution $\boldsymbol{\alpha}^*$ of the concave problem (7) and the optimal solution $\boldsymbol{w}^*$ of (6) are connected through the equality

$$\boldsymbol{w}^* = \frac{1}{2\lambda}\sum_{i=1}^n \alpha_i^*\phi(\boldsymbol{x}_i),$$

and $\boldsymbol{\alpha}^*$ can be found by setting the gradient of F_{KRR} to zero, to give

$$\boldsymbol{\alpha}^* = 2(I + \tfrac{1}{\lambda}K)^{-1}\boldsymbol{y}. \tag{8}$$

The value of the objective function at $\boldsymbol{\alpha}^*$ is then:

$$F_{KRR}(\boldsymbol{\alpha}^*) = \boldsymbol{y}^T(I + \tfrac{1}{\lambda}K)^{-1}\boldsymbol{y}, \tag{9}$$

and the resulting regression function is given by:

$$f(\boldsymbol{x}) = \frac{1}{2\lambda}\sum_{i=1}^n \alpha_i^* k(\boldsymbol{x}_i, \boldsymbol{x}). \tag{10}$$

2.3. A Convex Optimization Problem

KRR may be solved by solving the linear system $(I + \frac{K}{\lambda})\boldsymbol{\alpha} = 2\boldsymbol{y}$, at a cost of $O(n^3)$ operations. This might be prohibitive for large n, even more so if the matrix $I + \frac{K}{\lambda}$ does not fit into memory. To cope with this possible problem, we work with $\tilde{K}(\boldsymbol{\mu})$ (3) instead of the Gram matrix K. As we shall see, this not only makes it possible to avoid memory issues but it also allows us to set up a learning problem where both $\boldsymbol{\mu}$ and a regression function are sought for at once. This is very similar to the Multiple Kernel Learning paradigm (Rakotomamonjy et al., 2008) where one learns an optimal kernel along with the target function.

To set up the optimization problem we are interested in, we proceed in a way similar to (Rakotomamonjy et al., 2008). For $m = 1, \ldots, n$, define the Hilbert space $\tilde{\mathcal{H}}'_m$ as:

$$\tilde{\mathcal{H}}'_m := \left\{ f \in \tilde{\mathcal{H}}_m \,\middle|\, \frac{\|f\|_{\tilde{\mathcal{H}}_m}}{\mu_m} < \infty \right\}. \tag{11}$$

One can prove (Aronszajn, 1950) that $\tilde{\mathcal{H}} = \bigoplus \tilde{\mathcal{H}}'_m$ is the RKHS associated to $\tilde{k} = \sum \mu_m \tilde{k}_m$. Mimicking the reasoning of (Rakotomamonjy et al., 2008), our primal optimization problem reads:

$$\min_{\{f_m\},\boldsymbol{\mu}} \left\{ \lambda \sum_{m\in\mathcal{S}} \frac{1}{\mu_m} \|f_m\|^2_{\tilde{\mathcal{H}}'_m} + \sum_{i=1}^{n} (y_i - \sum_{m\in\mathcal{S}} f_m(\boldsymbol{x}_i))^2 \right\},$$
$$\text{s.t.} \sum_{m\in\mathcal{S}} \mu_m \leq n_1\,, \quad \mu_m \geq 0, \tag{12}$$

where n_1 is a parameter controlling the 1-norm of $\boldsymbol{\mu}$. As this problem is also convex in $\boldsymbol{\mu}$, using the earlier results on the KRR problem, (12) is equivalent to:

$$\min_{\boldsymbol{\mu}\geq 0} \left\{ \max_{\boldsymbol{\alpha}} \; \boldsymbol{y}^T\boldsymbol{\alpha} - \frac{1}{4\lambda}\boldsymbol{\alpha}^T(\lambda I + \tilde{K}(\boldsymbol{\mu}))\boldsymbol{\alpha} \right\}$$
$$= \min_{\boldsymbol{\mu}\geq 0} \left\{ \boldsymbol{y}^T(I + \tfrac{1}{\lambda}\tilde{K}(\boldsymbol{\mu}))^{-1}\boldsymbol{y} \right\} \text{s.t.} \sum_{m\in\mathcal{S}} \mu_m \leq n_1. \tag{13}$$

Finally, using the equivalence between Tikhonov and Ivanov regularization methods (Vasin, 1970), we obtain the *convex* and *smooth* optimization problem we focus on:

$$\min_{\boldsymbol{\mu}\geq 0} \left\{ F(\boldsymbol{\mu}) := \boldsymbol{y}^T(I + \tfrac{1}{\lambda}\tilde{K}(\boldsymbol{\mu}))^{-1}\boldsymbol{y} + \nu \sum_m \mu_m \right\}. \tag{14}$$

The regression function $\tilde{f}$ is derived using (1), a minimizer $\boldsymbol{\mu}^*$ of the latter problem and the accompanying weight vector $\boldsymbol{\alpha}^*$ such that

$$\boldsymbol{\alpha}^* = 2\left(I + \tfrac{1}{\lambda}\tilde{K}(\boldsymbol{\mu}^*)\right)^{-1}\boldsymbol{y}, \tag{15}$$

(obtained adapting (8) to the case $K = K(\boldsymbol{\mu}^*)$). We have:

$$\tilde{f}(\boldsymbol{x}) = \frac{1}{2\lambda}\sum_{i=1}^{n} \alpha_i^* \tilde{k}(\boldsymbol{x}_i, \boldsymbol{x}) = \frac{1}{2\lambda}\sum_{m\in\mathcal{S}} \mu_m^* \sum_{i=1}^{n} \alpha_i^* \tilde{k}_m(\boldsymbol{x}_i, \boldsymbol{x})$$
$$= \frac{1}{2\lambda}\sum_{m\in\mathcal{S}} \tilde{\alpha}_m^* k(\boldsymbol{x}_m, \boldsymbol{x}), \tag{16}$$

$$\text{where} \qquad \tilde{\alpha}_m^* := \mu_m^* \frac{\boldsymbol{c}_m^\top \boldsymbol{\alpha}^*}{\sqrt{k(\boldsymbol{x}_m, \boldsymbol{x}_m)}}. \tag{17}$$

3. Solving the problem

We now introduce a new stochastic optimization procedure to solve (14). It implements a coordinate descent strategy with step sizes that use second-order information.

3.1. A Second-Order Stochastic Coordinate Descent

Problem (14) is a constrained minimization based on the differentiable and convex objective function F. Usual convex optimization methods (such as projected gradient descent, proximal methods) may be employed to solve this problem, but they may be too computationally expensive if n is very large, which is essentially due to a suboptimal exploitation of the parametrization of the problem. Instead, the optimization strategy we propose is specifically tailored to take advantage of the parametrization of $\tilde{K}(\boldsymbol{\mu})$.

Algorithm 1 depicts our stochastic descent method, inspired by (Nesterov, 2010). At each iteration, a randomly chosen coordinate of $\boldsymbol{\mu}$ is updated via a Newton step. This method has two essential features: i) using coordinate-wise updates of $\boldsymbol{\mu}$ involves only partial derivatives which can be easily computed and ii) the stochastic approach ensures a reduced memory cost while still guaranteeing convergence.

Algorithm 1 Stochastic Coordinate Newton Descent

Input: $\boldsymbol{\mu}^0$ random.
repeat
Choose coordinate m_k uniformly at random in $\mathcal{S}$.
Update : $\mu_m^{k+1} = \mu_m^k$ if $m \neq m_k$ and

$$\mu_{m_k}^{k+1} = \operatorname*{argmin}_{v\geq 0} \frac{\partial F(\boldsymbol{\mu}^k)}{\partial \mu_{m_k}}(v - \mu_{m_k}^k) + \frac{1}{2}\frac{\partial^2 F(\boldsymbol{\mu}^k)}{\partial \mu_{m_k}^2}(v - \mu_{m_k}^k)^2, \tag{18}$$

until $F(\boldsymbol{\mu}^k) - F(\boldsymbol{\mu}^{k-M}) < \epsilon F(\boldsymbol{\mu}^{k-M})$

Notice that the Stochastic Coordinate Newton Descent (SCND) is similar to the algorithm proposed in (Nesterov, 2010), except that we replace the Lipschitz constants by the second-order partial derivatives $\frac{\partial^2 F(\mu^k)}{\partial \mu_{m_k}^2}$. Thus, we replace a constant step-size gradient descent by a the Newton-step in (18), which allows us to make larger steps.

We show that for the function F in (14), SCND does provably converge to a minimizer of Problem (14). First, we rewrite (18) as a Newton step and compute the partial derivatives:

Proposition 1. *Eq. (18) is equivalent to*

$$\mu_{m_k}^{k+1} = \begin{cases} \left(\mu_{m_k}^k - \frac{\partial F(\mu^k)}{\partial \mu_{m_k}} \Big/ \frac{\partial^2 F(\boldsymbol{\mu}^k)}{\partial \mu_{m_k}^2}\right)_+ & \text{if } \frac{\partial^2 F(\boldsymbol{\mu}^k)}{\partial \mu_{m_k}^2} \neq 0 \\ 0 \text{ otherwise.} \end{cases} \tag{19}$$

Proof. (19) gives the optimality conditions for (18). □

Proposition 2. *The partial derivatives $\frac{\partial^p F(\boldsymbol{\mu})}{\partial \mu_m^p}$ are:*

$$\frac{\partial F(\boldsymbol{\mu})}{\partial \mu_m} = -\lambda(\boldsymbol{y}^\top \tilde{K}_{\lambda,\boldsymbol{\mu}}^{-1}\boldsymbol{c}_m)^2 + \nu, \tag{20}$$
$$\frac{\partial^p F(\boldsymbol{\mu})}{\partial \mu_m^p} = (-1)^p p!\lambda(\boldsymbol{y}^\top \tilde{K}_{\lambda,\boldsymbol{\mu}}^{-1}\boldsymbol{c}_m)^2(\boldsymbol{c}_m^\top \tilde{K}_{\lambda,\boldsymbol{\mu}}^{-1}\boldsymbol{c}_m)^{p-1},$$
$$\text{with } p \geq 2 \text{ and } \tilde{K}_{\lambda,\boldsymbol{\mu}}^{-1} := (\lambda I + \tilde{K}(\boldsymbol{\mu}))^{-1}. \tag{21}$$

Proof. Easy but tedious calculations give the results. □

Theorem 1 (Convergence)**.** *For any sequence $\{m_k\}_k$, the sequence $\{F(\boldsymbol{\mu}^k)\}_k$ verifies:*

(a) $\forall k,\ F(\boldsymbol{\mu}^{k+1}) \leq F(\boldsymbol{\mu}^k)$.

(b) $\lim_{k\to\infty} F(\boldsymbol{\mu}^k) = \min_{\boldsymbol{\mu}\geq 0} F(\boldsymbol{\mu})$.

Moreover, if there exists a minimizer $\boldsymbol{\mu}^*$ of F such that the Hessian $\nabla^2 F(\boldsymbol{\mu}^*)$ is positive definite then:

(c) $\boldsymbol{\mu}^*$ is the unique minimizer of F. The sequence $\{\boldsymbol{\mu}^k\}$ converges to $\boldsymbol{\mu}^*$: $||\boldsymbol{\mu}^k - \boldsymbol{\mu}^*|| \to 0$.

Sketch of proof. (a) Using that $\frac{\partial^3 F(\boldsymbol{\mu})}{\partial \mu_m^3} \leq 0$ (see (20)), one shows that the Taylor series truncated to the second order: $\boldsymbol{v} \to F(\boldsymbol{\mu}) + \frac{\partial F(\boldsymbol{\mu})}{\partial \mu_m}(\boldsymbol{v}_m - \boldsymbol{\mu}_m) + \frac{1}{2}\frac{\partial^2 F(\boldsymbol{\mu})}{\partial \mu_m^2}(\boldsymbol{v}_m - \boldsymbol{\mu}_m)^2$, is a quadratic upper-bound of F that matches F and ∇F at point $\boldsymbol{\mu}$ (for any fixed m and $\boldsymbol{\mu}$). From this, the update formula (18) yields $F(\boldsymbol{\mu}^{k+1}) \leq F(\boldsymbol{\mu}^k)$.

(b) First note that $||\boldsymbol{\mu}^k|| \leq F(\boldsymbol{\mu}^0)$ and extract a converging subsequence $\{\boldsymbol{\mu}^{\phi(k)}\}$. Denote the limit by $\hat{\boldsymbol{\mu}}$. Separating the cases where $\frac{\partial^2 F(\hat{\boldsymbol{\mu}})}{\partial \mu_m^2}$ is zero or not, one shows that $\hat{\boldsymbol{\mu}}$ satisfies the optimality conditions: $\langle \nabla F(\hat{\boldsymbol{\mu}}), \boldsymbol{v} - \hat{\boldsymbol{\mu}}\rangle \geq 0,\ \forall \boldsymbol{v} \geq 0$. Thus $\hat{\boldsymbol{\mu}}$ is a minimizer of F and we have $\lim F(\boldsymbol{\mu}^k) = \lim F(\boldsymbol{\mu}^{\phi(k)}) = F(\hat{\boldsymbol{\mu}}) = \min_{\boldsymbol{\mu}\geq 0} F(\boldsymbol{\mu})$.

(c) is standard in convex optimization. □

3.2. Iterative Updates

One may notice that the computations of the derivatives (20), as well as the computation of $\boldsymbol{\alpha}^*$, depend on $\tilde{K}^{-1}_{\lambda,\boldsymbol{\mu}}$. Moreover, the dependency in $\boldsymbol{\mu}$, for all those quantities, only lies in $\tilde{K}^{-1}_{\lambda,\boldsymbol{\mu}}$. Thus, a special care need be taken on how $\tilde{K}^{-1}_{\lambda,\boldsymbol{\mu}}$ is stored and updated throughout.

Let $\mathcal{S}^+_{\boldsymbol{\mu}} = \{m \in \mathcal{S} | \mu_m > 0\}$ and $m_0 = \|\boldsymbol{\mu}\|_0 = |\mathcal{S}^+_{\boldsymbol{\mu}}|$. Let $C = [\boldsymbol{c}_{i_1} \cdots \boldsymbol{c}_{i_{m_0}}]$ be the concatenation of the $\boldsymbol{c}_{i_j}$'s, for $i_j \in \mathcal{S}^+_{\boldsymbol{\mu}}$ and D the diagonal matrix with diagonal elements μ_{i_j}, for $i_j \in \mathcal{S}^+_{\boldsymbol{\mu}}$. Remark that throughout the iterations the sizes of C and D may vary. Given (21) and using Woodbury formula (Theorem 2, Appendix), we have:

$$\tilde{K}^{-1}_{\lambda,\boldsymbol{\mu}} = \left(\lambda I + CDC^\top\right)^{-1} = \frac{1}{\lambda} I - \frac{1}{\lambda^2} CGC^\top \qquad (22)$$

$$\text{with} \qquad G := \left(D^{-1} + \frac{1}{\lambda} C^\top C\right)^{-1}. \qquad (23)$$

Note that G is a square matrix of order m_0 and that an update on $\boldsymbol{\mu}$ will require an update on G. Even though updating G^{-1}, i.e. $D^{-1} + \frac{1}{\lambda}C^\top C$, is trivial, it is more efficient to directly store and update G. This is what we describe now.

At each iteration, only one coordinate of $\boldsymbol{\mu}$ is updated. Let p be the index of the updated coordinate, $\boldsymbol{\mu}_{\text{old}}$, C_{old}, D_{old} and G_{old}, the vectors and matrices before the update and $\boldsymbol{\mu}_{\text{new}}$, C_{new}, D_{new} and G_{new} the updated matrices/vectors. Let also $\boldsymbol{e}_p$ bethe vector whose pth coordinate is 1 while other coordinates are 0. We encounter four different cases.

Case 1: $\mu_p^{\text{old}} = 0$ and $\mu_p^{\text{new}} = 0$. No update needed:

$$G_{\text{new}} = G_{\text{old}}. \qquad (24)$$

Case 2: $\mu_p^{\text{old}} \neq 0$ and $\mu_p^{\text{new}} \neq 0$. Here, $C_{\text{old}} = C_{\text{new}}$ and

$$D^{-1}_{\text{new}} = D^{-1}_{\text{old}} + \Delta_p \boldsymbol{e}_p \boldsymbol{e}_p^\top, \quad \text{where} \quad \Delta_p := \frac{1}{\mu_p^{\text{new}}} - \frac{1}{\mu_p^{\text{old}}}.$$

Then, using Woodbury formula, we have:

$$G_{\text{new}} = \left(G^{-1}_{\text{old}} + \Delta_p \boldsymbol{e}_p \boldsymbol{e}_p^\top\right)^{-1} = G_{\text{old}} - \frac{\Delta_p}{1 + \Delta_p g_{pp}} \boldsymbol{g}_p \boldsymbol{g}_p^\top, \qquad (25)$$

with g_{pp} the (p,p)th entry of G_{old} and $\boldsymbol{g}_p$ its pth column.

Case 3: $\mu_p^{\text{old}} \neq 0$ and $\mu_p^{\text{new}} = 0$. Here, $\mathcal{S}^+_{\boldsymbol{\mu}_{\text{new}}} = \mathcal{S}^+_{\boldsymbol{\mu}_{\text{old}}} \setminus \{p\}$. It follows that we have to remove $\boldsymbol{c}_p$ from C_{old} to have C_{new}. To get G_{new}, we may consider the previous update formula when $\mu_p^{\text{new}} \to 0$ (that is, when $\Delta_p \to +\infty$). Note that we can use the previous formula because $\mu_p \mapsto \tilde{K}^{-1}_{\lambda,\boldsymbol{\mu}}$ is well-defined and continuous at 0. Thus, as $\lim_{\mu_p^{\text{new}}\to 0} \frac{\Delta_p}{1+\Delta_p g_{pp}} = \frac{1}{g_{pp}}$, we have:

$$G_{\text{new}} = \left(G_{\text{old}} - \frac{1}{g_{pp}} \boldsymbol{g}_p \boldsymbol{g}_p^\top\right)_{\setminus\{p\}}, \qquad (26)$$

where $A_{\setminus\{p\}}$ denotes the matrix A from which the pth column and pth row have been removed.

Case 4: $\mu_p^{\text{old}} = 0$ and $\mu_p^{\text{new}} \neq 0$. We have $C_{\text{new}} = [C_{\text{old}}\ \boldsymbol{c}_p]$. Using (23), it follows that

$$\begin{aligned} G_{\text{new}} &= \begin{pmatrix} D^{-1}_{\text{old}} + \frac{1}{\lambda} C^\top_{\text{old}} C_{\text{old}} & \frac{1}{\lambda} C^\top_{\text{old}} \boldsymbol{c}_p \\ \frac{1}{\lambda} \boldsymbol{c}_p^\top C_{\text{old}} & \frac{1}{\mu_p^{\text{new}}} + \frac{1}{\lambda} \boldsymbol{c}_p^\top \boldsymbol{c}_p \end{pmatrix}^{-1} \\ &= \begin{pmatrix} G^{-1}_{\text{old}} & \frac{1}{\lambda} C^\top_{\text{old}} \boldsymbol{c}_p \\ \frac{1}{\lambda} \boldsymbol{c}_p^\top C_{\text{old}} & \frac{1}{\mu_p^{\text{new}}} + \frac{1}{\lambda} \boldsymbol{c}_p^\top \boldsymbol{c}_p \end{pmatrix}^{-1} \\ &= \begin{pmatrix} A & \boldsymbol{v} \\ \boldsymbol{v}^\top & s \end{pmatrix}, \end{aligned}$$

where, using the block-matrix inversion formula of Theorem 3 (Appendix), we have:

$$\begin{aligned} s &= \left(\frac{1}{\mu_p^{\text{new}}} + \frac{1}{\lambda} \boldsymbol{c}_p^\top \boldsymbol{c}_p - \frac{1}{\lambda^2} \boldsymbol{c}_p^\top C_{\text{old}} G_{\text{old}} C^\top_{\text{old}} \boldsymbol{c}_p\right)^{-1} \\ \boldsymbol{v} &= -\frac{s}{\lambda} G_{\text{old}} C^\top_{\text{old}} \boldsymbol{c}_p \qquad (27) \\ \boldsymbol{A} &= G_{\text{old}} + \frac{1}{s} \boldsymbol{v}\boldsymbol{v}^\top. \end{aligned}$$

Algorithm 2 SLKL: Stochastic Low-Rank Kernel Learning

inputs: $\mathcal{L} := \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $\nu > 0$, $M > 0$, $\epsilon > 0$.
outputs: $\boldsymbol{\mu}$, G and C (yield $(\lambda I + K(\mu))^{-1}$ from (22)).

initialization: $\boldsymbol{\mu}^{(0)} = \mathbf{0}$.
repeat
Choose coordinate m_k uniformly at random in $\mathcal{S}$.
Update $\boldsymbol{\mu}^{(k)}$ according to (19), by changing only the m_k-th coordinate $\mu^k_{m_k}$ of $\boldsymbol{\mu}^{(k)}$:

- compute the second order derivative

$$h = \lambda(\boldsymbol{y}^\top \tilde{K}^{-1}_{\lambda,\boldsymbol{\mu}} \boldsymbol{c}_{m_k})^2 (\boldsymbol{c}_{m_k}^\top \tilde{K}^{-1}_{\lambda,\boldsymbol{\mu}} \boldsymbol{c}_{m_k}) \; ;$$

- **if** $h > 0$ **then**

$$\mu^{(k+1)}_{m_k} = \max\left(0, \mu^{(k)}_{m_k} + \frac{\lambda(\boldsymbol{y}^\top \tilde{K}^{-1}_{\lambda,\boldsymbol{\mu}} \boldsymbol{c}_{m_k})^2 - \nu}{h}\right);$$

else $\mu^{(k+1)}_{m_k} = 0$.

Update $G^{(k)}$ and $C^{(k)}$ according to (24)-(27).
until $F(\boldsymbol{\mu}^k) - F(\boldsymbol{\mu}^{k-M}) < \epsilon F(\boldsymbol{\mu}^{k-M})$

Complete learning algorithm. Algorithm 2 depicts the full Stochastic Low-Rank Kernel Learning algorithm (SLKL), which recollects all the pieces just described.

4. Analysis

Here, we discuss the relation between λ and ν and we argue that there is no need to keep both hyperparameters. In addition, we provide a short analysis on the runtime complexity of our learning procedure.

4.1. Pivotal Hyperparameter $\lambda\nu$

First recall that we are interested in the minimizer $\boldsymbol{\mu}^*_{\lambda,\nu}$ of constrained optimization problem (14), i.e.:

$$\boldsymbol{\mu}^*_{\lambda,\nu} = \operatorname*{argmin}_{\boldsymbol{\mu} \geq 0} F_{\lambda,\nu}(\boldsymbol{\mu}), \tag{28}$$

where, for the sake of clarity, we purposely show the dependence on λ and ν of the objective function $F_{\lambda,\nu}$

$$F_{\lambda,\nu}(\boldsymbol{\mu}) = \boldsymbol{y}^\top \left(I + \tilde{K}\left(\tfrac{\boldsymbol{\mu}}{\lambda}\right)\right)^{-1} \boldsymbol{y} + \lambda\nu \sum_m \tfrac{\mu_m}{\lambda}, \tag{29}$$

We may name $\boldsymbol{\alpha}^*_{\lambda,\nu}$, $\tilde{\boldsymbol{\alpha}}^*_{\lambda,\nu}$ the weight vectors associated with $\boldsymbol{\mu}^*_{\lambda,\nu}$ (see (15) and (17)). We have the following:

Proposition 3. *Let $\lambda, \nu, \lambda', \nu'$ be strictly positive real numbers. If $\lambda\nu = \lambda'\nu'$ then*

$$\boldsymbol{\mu}^*_{\lambda',\nu'} = \tfrac{\lambda'}{\lambda}\boldsymbol{\mu}^*_{\lambda,\nu}, \quad \textit{and} \quad \tilde{f}_{\lambda,\nu} = \tilde{f}_{\lambda',\nu'}.$$

As a direct consequence:

$$\forall \lambda, \nu \geq 0, \; \tilde{f}_{\lambda,\nu} = \tilde{f}_{1,\lambda\nu}.$$

Proof. Suppose that we know $\boldsymbol{\mu}^*_{\lambda,\nu}$. Given the definition (29) of $F_{\lambda,\nu}$ and using $\lambda\nu = \lambda'\nu'$, we have

$$F_{\lambda,\nu}(\boldsymbol{\mu}) = F_{\lambda',\nu'}\left(\tfrac{\lambda'}{\lambda}\boldsymbol{\mu}\right)$$

Since the only constraint of problem (28) is the nonnegativity of the components of $\boldsymbol{\mu}$, it directly follows that $\lambda'\boldsymbol{\mu}^*_{\lambda,\nu}/\lambda$ is a minimizer of $F_{\lambda',\nu'}$ (under these constraints), hence $\boldsymbol{\mu}^*_{\lambda',\nu'} = \lambda'\boldsymbol{\mu}^*_{\lambda,\nu}/\lambda$.

To show $\tilde{f}_{\lambda,\nu} = \tilde{f}_{\lambda',\nu'}$, it suffices to observe that, according to the way $\boldsymbol{\alpha}^*_{\lambda,\nu}$ is defined (cf. (15)),

$$\begin{aligned}\boldsymbol{\alpha}^*_{\lambda',\nu'} &= 2\left(I + K\left(\tfrac{\boldsymbol{\mu}^*_{\lambda',\nu'}}{\lambda'}\right)\right)^{-1}\boldsymbol{y} \\ &= 2\left(I + K\left(\tfrac{\lambda'}{\lambda}\tfrac{\boldsymbol{\mu}^*_{\lambda,\nu}}{\lambda'}\right)\right)^{-1}\boldsymbol{y} = \boldsymbol{\alpha}^*_{\lambda,\nu},\end{aligned}$$

and, thus, $\tilde{\boldsymbol{\alpha}}^*_{\lambda',\nu'} = \lambda'\tilde{\boldsymbol{\alpha}}^*_{\lambda,\nu}/\lambda$. The definition (16) of $\tilde{f}_{\lambda,\nu}$ then gives $\tilde{f}_{\lambda,\nu} = \tilde{f}_{\lambda',\nu'}$, which entails $\tilde{f}_{\lambda,\nu} = \tilde{f}_{1,\lambda\nu}$. □

This proposition has two nice consequences. First, it says that the pivotal hyperparameter is actually the product $\lambda\nu$: this *is* the quantity that parametrizes the learning problem (not λ or ν, seen independently). Thus, the set of regression functions, defined by the λ and ν hyperparameter space, can be described by exploring the set of vectors $(\boldsymbol{\mu}^*_{1,\nu})_{\nu>0}$, which only depends on a single parameter. Second, considering $(\boldsymbol{\mu}^*_{1,\nu})_{\nu>0}$ allows us to work with the family of objective functions $(F_{1,\nu})_{\nu>0}$, which are well-conditioned numerically as the hyperparameter λ is set to 1.

4.2. Runtime Complexity and Memory Usage

For the present analysis, let us assume that we pre-compute the M (randomly) selected columns $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_M$. If a is the cost of computing a column $\boldsymbol{c}_m$, the pre-computation has a cost of $O(Ma)$ and has a memory usage of $O(nM)$.

At each iteration, we have to compute the first and second-order derivatives of the objective function, as well as its value and the weight vector $\boldsymbol{\alpha}$. Using (22), (20), (14) and (15), one can show that those operations have a complexity of $O(nm_0)$ if m_0 is the zero-norm of $\boldsymbol{\mu}$.

Besides, in addition to C, we need to store G for a memory cost of $O(m_0^2)$. Overall, if we denote the number of iterations by k, the algorithm has a memory cost of $O(nM + m_0^2)$ and a complexity of $O(knm_0 + Ma)$.

If memory is a critical issue, one may prefer to compute the columns $\boldsymbol{c}_m$ on-the-fly and m_0 columns need to be stored instead of M (this might be a substantial saving in terms of memory as can be seen in the next section). This improvement in term of memory usage implies an additive cost in the runtime complexity. In the worst case, we have

to compute a new column $\boldsymbol{c}$ at each iteration. The resulting memory requirement scales as $O(nm_0 + m_0^2)$ and the runtime complexity varies as $O(k(nm_0 + a))$.

5. Numerical Simulations

We now present results from various numerical experiments, for which we describe the datasets and the protocol used. We study the influence of the different parameters of our learning approach on the results and compare the performance of our algorithm to that of related methods.

5.1. Setup

First, we use a toy dataset (denoted by *sinc*) to better understand the role and influence of the parameters. It consists in regressing the cardinal sine of the two-norm (i.e. $\boldsymbol{x} \mapsto \sin(\|\boldsymbol{x}\|)/\|\boldsymbol{x}\|$) of random two-dimensional points, each drawn uniformly between -5 and $+5$. In order to have a better idea on how the solutions may or may not over-fit the training data, we add some white Gaussian noise on the target variable of the randomly picked 1000 training points (with a 10 dB signal-to-noise ratio). The test set is made of 1000 non-noisy independent instance/target pairs.

We then assess our method on two UCI datasets: Abalone (*abalone*) and Boston Housing (*boston*), using the same normalizations, Gaussian kernel parameters (σ denotes the kernel width) and data partition as in (Smola & Schölkopf, 2000). The United States Postal Service (*USPS*) dataset is used with the same setting as in (Williams & Seeger, 2001). Finally, the Modified National Institute of Standards and Technology (*MNIST*) dataset is used with the same preprocessing as in (Maji & Malik, 2009). Table 1 summarizes the characteristics of all the datasets we used.

Table 1. Datasets used for the experiments.

dataset	**#features**	**#train** (n)	**#test**	$\boldsymbol{\sigma^2}$
sinc	2	1000	1000	1
abalone	10	3000	1177	2.5
boston	13	350	156	3.25
USPS	256	7291	2007	64
MNIST	2172	60000	10000	4

As displayed in Algorithm 1, at each iteration $k > M$, we check if $F(\boldsymbol{\mu}^k) - F(\boldsymbol{\mu}^{k-M}) < \epsilon F(\boldsymbol{\mu}^{k-M})$ holds. If so, we stop the optimization process. ϵ thus controls our stopping criterion. In the experiments, we set $\epsilon = 10^{-4}$ unless otherwise stated and we set λ to 1 for all the experiments and we run simulations for various values of ν and M. In order to assess the variability incurred by the stochastic nature of our learning algorithm, we run each experiment 20 times.

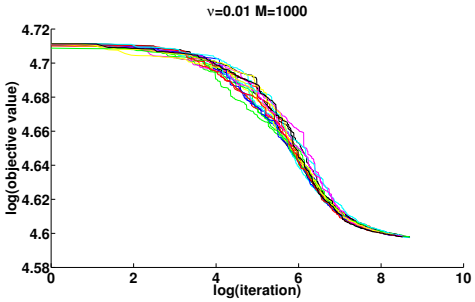

Figure 1. Evolution of the objective during the optimization process for the *sinc* dataset with $\nu = 0.01$, $M = 1000$ (for 20 runs).

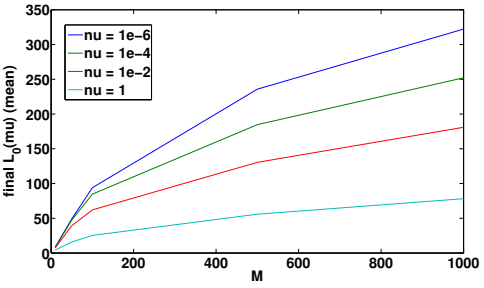

Figure 2. Zero-norm of the optimal $\boldsymbol{\mu}^*$ as a function of M for different values of ν for the *sinc* dataset (averaged on 20 runs).

5.2. Influence of the parameters

5.2.1. Evolution of the objective

We have established (Section 3) the convergence of our optimization procedure, under mild conditions. A question that we have not tackled yet is to evaluate its convergence rate. Figure 1 plots the evolution of the objective function on the *sinc* dataset. We observe that the evolutions of the objective function are impressively similar among the different runs. This empirically tends to assert that it is relevant to look for theoretical results on the convergence rate.

A question left for future work is the impact of the random selection of the set of columns $\mathcal{S}$ on the reached solution.

5.2.2. Zero-norm of $\boldsymbol{\mu}$

As shown in Section 4.2, both memory usage and the complexity of the algorithm depend on m_0. Thus, it is interesting to take a closer look at how this quantity evolves. Figure 2 and 3 experimentally point out two things. On the one hand, the number of active components $m_0 = \|\boldsymbol{\mu}\|_0$ remains significantly smaller than M. In other words, as long as the regularization parameter is well-chosen, we never have to store all of the $\boldsymbol{c}_m$ at the same time. On the other hand, the solution $\boldsymbol{\mu}^*$ is sparse and $\|\boldsymbol{\mu}^*\|_0$ grows with M and diminishes with ν. A theoretical study on the dependence of $\boldsymbol{\mu}^*$ and m_0 in M and ν, left for future work, would

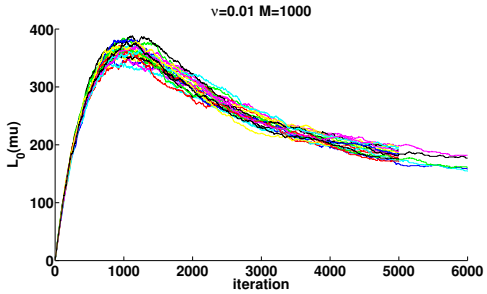

Figure 3. Evolution of the zero-norm of $\boldsymbol{\mu}$ (m_0) with the iterations for the *sinc* dataset with $\nu = 0.01$, $M = 1000$ (20 runs).

be all the more interesting since sparsity is the cornerstone on which the scalability of our algorithm depends.

5.3. Comparison to other methods

This section aims at giving a hint on how our method performs on regression tasks. To do so, we compare the Mean Square Error (over the test set). In addition to our Stochastic Low-Rank Kernel Learning method (*SLKL*), we solve the problem with the standard Kernel Ridge Regression method, using the n training data (*KRRn*) and using only M training data (*KRRM*). We also evaluate the performance of the KRR method, using the kernel obtained with uniform weights on the M rank-1 approximations selected for *SLKL* (*Unif*). The results are displayed in Table 2, where the bold font indicates the best low-rank method (*KRRM, Unif* or *SLKL*) for each experiment.

Table 2 confirms that optimizing the weight vector $\boldsymbol{\mu}$ is decisive as our results dramatically outperform those of *Unif*. As long as $M < n$, our method also outperforms *KRRM*. The explanation probably lies in the fact that our approximations keep information about similarities between the M selected points and the $n - M$ others. Furthermore, our method *SLKL* achieves comparable performances (or even better on *abalone*) than *KRRn*, while finding sparse solutions. Compared to the approach from (Smola & Schölkopf, 2000), we seem to achieve lower test error on the *boston* dataset even for $M = 128$. On the *abalone* dataset, this method outperforms ours for every M we tried.

Finally, we also compare the results we obtain on the *USPS* dataset with the ones obtained in (Williams & Seeger, 2001) (*Nyst*). As it consists in a classification task, we actually perform a regression on the labels to adapt our method, which is known to be equivalent to solving Fisher Discriminant Analysis (Duda & Hart, 1973). The performance achieved by *Nyst* outperforms ours. However, one may argue that the performance have a same order of magnitude and note that the *Nyst* approach focuses on the classification task, while ours was designed for regression.

Table 3. Number of errors and standard deviation on the test set (2007 examples) of the USPS dataset.

M	64	256	1024
Nyst	101.3 ± 22.9	34.5 ± 3.0	35.9 ± 2.0
SLKL	76.3 ± 9.9	47.6 ± 3.1	41.5 ± 3.9
m_0	61	210	515

5.4. Large-scale dataset

To assess the scalability of our method, we ran experiments on the larger handwritten digits *MNIST* dataset, whose training set is made of 60000 examples. We used a Gaussian kernel computed over histograms of oriented gradients as in (Maji & Malik, 2009), in a "one versus all" setting. For $M = 1000$, we obtained classification error rates around 2% over the test set, which do not compete with state-of-the-art results but achieve reasonable performance, considering that we use only a small part of the data (cf. the size of M) and that our method was designed for regression.

Although our method overcomes memory usage issues for such large-scale problems, it still is computationally intensive. In fact, a large number of iterations is spent picking coordinates whose associated weight remains at 0. Though those iterations do not induce any update, they do require computing the associated Gram matrix column (which is not stored as it does not weigh in the conic combination) as well as the derivatives of the objective function. The main focus of our future work is to avoid those computations, using e.g. techniques such as shrinkage (Hsieh et al., 2008).

6. Conclusion

We have presented an original kernel-based learning procedure for regression. The main features of our contribution are the use of a conical combination of data-based kernels and the derivation of a stochastic convex optimization procedure, that acts coordinate-wise and makes use of second-order information. We provide theoretical convergence guarantees for this optimization procedure, we depict the behavior of our learning procedure and illustrate its effectiveness through a number of numerical experiments carried out on several benchmark datasets.

The present work naturally raises several questions. Among them, we may pinpoint that of being able to establish precise rate of convergence for the stochastic optimization procedure and that of generalizing our approach to the use of several kernels. Establishing data-dependent generalization bounds taking advantage of either the one-norm constraint on $\boldsymbol{\mu}$ or the size M of the kernel combination is of primary importance to us. The connection established between the one-norm hyperparameter ν and the ridge pa-

Table 2. Mean square error with standard deviation measured on three regression tasks.

	sinc			*boston*			*abalone*		
M	256	512	1000	128	256	350	512	1024	3000
KRRn	0.009 ± 09			10.17 ± 0			6.91 ± 0		
KRRM	0.0146 $\pm 1e^{-3}$	0.0124 $\pm 7e^{-4}$	$\mathbf{0.0099}$ ± 0	33.27 ± 7.8	16.89 ± 3.27	$\mathbf{10.17}$ ± 0	6.14 ± 0.25	5.51 ± 0.09	5.25 ± 0
Unif	0.0124 $\pm 1e^{-4}$	0.0124 $\pm 3e^{-5}$	0.0124 ± 0	149.7 ± 5.57	147.84 ± 2.24	147.72 ± 0	10.04 ± 0.17	9.96 ± 0.06	9.99 ± 0
SLKL	$\mathbf{0.0106}$ $\pm 4e^{-4}$	$\mathbf{0.0103}$ $\pm 2e^{-4}$	0.0104 $\pm 1e^{-4}$	$\mathbf{20.17}$ ± 2.3	$\mathbf{13.1}$ ± 0.87	11.43 ± 0.06	$\mathbf{5.04}$ ± 0.08	$\mathbf{4.94}$ ± 0.03	$\mathbf{4.95}$ ± 0.004
m_0	83	108	139	108	161	184	159	191	253

rameter λ, in section 4, seems interesting and may be witnessed in (Rakotomamonjy et al., 2008). Although not been mentioned so far, there might be connections between our modeling strategy and boosting/leveraging-based optimization procedures. Finally, we plan on generalizing our approach to other kernel methods, noting that rank-1 update formulas as those proposed here can possibly be exhibited even for problems with no closed-form solution.

Acknowledgments

This work is partially supported by the IST Program of the European Community, under the FP7 Pascal 2 Network of Excellence (ICT-216886-NOE) and by the ANR project LAMPADA (ANR-09-EMER-007).

A. Matrix Inversion Formulas

Theorem 2. (Woodbury matrix inversion formula (Woodbury, 1950)) *Let n and m be positive integers, $A \in \mathbb{R}^{n\times n}$ and $C \in \mathbb{R}^{m\times m}$ be non-singular matrices and let $U \in \mathbb{R}^{n\times m}$ and $V \in \mathbb{R}^{m\times n}$ be two matrices. If $C^{-1}+VA^{-1}U$ is non-singular then so is $A+UCV$ and:*

$$(A+UCV)^{-1} = A^{-1}-A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}.$$

Theorem 3. (Matrix inversion with added column) *Given m, integer and $M \in \mathbb{R}^{(n+1)\times(n+1)}$ partitioned as:*

$$M = \begin{pmatrix} A & \boldsymbol{b} \\ \boldsymbol{b}^\top & c \end{pmatrix}, \quad \text{where } A \in \mathbb{R}^{n\times n}, \boldsymbol{b} \in \mathbb{R}^n \text{ and } c \in \mathbb{R}.$$

If A is non-singular and $c - \boldsymbol{b}^\top A^{-1}\boldsymbol{b} \neq 0$, then M is non-singular and the inverse of M is given by

$$M^{-1} = \begin{pmatrix} A^{-1} + \frac{1}{k}A^{-1}\boldsymbol{b}\boldsymbol{b}^\top A^{-1} & -\frac{1}{k}A^{-1}\boldsymbol{b} \\ -\frac{1}{k}\boldsymbol{b}^\top A^{-1} & \frac{1}{k} \end{pmatrix}, \quad (30)$$

where $k = c - \boldsymbol{b}^\top A^{-1}\boldsymbol{b}$.

References

Aronszajn, N. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, May 1950.

Drineas, P. and Mahoney, M. On the nyström method for approximating a gram matrix for improved kernel-based learning. *J. of Machine Learning Research*, 6:2153–2175, Dec. 2005.

Duda, Richard O. and Hart, Peter E. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

Hoerl, A. and Kennard, R. Ridge regression: applications to nonorthogonal problems. *Technometrics*, 12(1):69–82, 1970.

Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. Sathiya, and Sundararajan, S. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 408–415, 2008.

Kumar, S., Mohri, M., and Talwalkar, A. Ensemble nyström method. In *Advances in Neural Information Processing Systems 22*, pp. 1060–1068, 2009.

Maji, S. and Malik, J. Fast and accurate digit classification. Technical report, EECS Department, UC Berkeley, 2009.

Nesterov, Y. Efficiency of coordinate descent methods on huge-scale optimization problems. Core discussion papers, 2010.

Rakotomamonjy, A., Bach, F., Canu, S., and Grandvalet, Y. Simplemkl. *J. of Machine Learning Research*, 9:2491–2521, 2008.

Schölkopf, B., Mika, S., Burges, C. J. C., Knirsch, P., Müller, K.-R., Rätsch, G., and Smola, A. J. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, September 1999.

Smola, A. J. and Schölkopf, B. Sparse greedy matrix approximation for machine learning. In *International Conference on Machine Learning*, pp. 911–918, 2000.

Suykens, Johan A. K., Gestel, Tony Van, Brabanter, Jos De, Moor, Bart De, and Vandewalle, Joos. *Least Squares Support Vector Machines*, chapter 6, pp. 173–182. World Scientific, 2002.

Vasin, V. V. Relationship of several variational methods for the approximate solution of ill-posed problems. *Mathematical Notes*, 7(3):161–166, 1970.

Williams, C. and Seeger, M. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pp. 682–688. MIT Press, 2001.

Woodbury, M. A. Inverting modified matrices. Technical report, Statistical Research Group, Princeton University, 1950.